

A simplified approach to establishing the impact of software source code changes on requirements specifications

Fredrick Mugambi Muthengi¹, David Muchangi Mugo¹, Stephen Makau Mutua², Faith Mueni Musyoka¹

¹Department of Computing and Information Technology, Faculty of Applied Sciences, University of Embu, Embu, Kenya

²Department of Computer Science, School of Computing and Informatics, Meru University of Science and Technology, Meru, Kenya

Article Info

Article history:

Received May 24, 2024

Revised Sep 3, 2024

Accepted Sep 28, 2024

Keywords:

Association

Change impact

Changed method

Functional requirement

Method name

ABSTRACT

This paper proposes an improved approach to establishing the impact of source code changes in software features. An association of the methods affected by the changes made and functional requirements of the software reveals the likely impact of the changes made in the software. Changes in source code are exhibited in the operational behaviour of the software. Functional requirements and source code artefacts play a key role in assessing the impact of the changes made. In this study, we investigated the possibility of establishing the association between the changed methods and the functional requirements. The study found out that changes made in the methods can be mapped to the functional requirements that the methods are implementing. The motivation in this endeavour was to assess the impacted software requirements which translate to the likely software features affected by the changes made in the software. With the intent of the software users being seen in the requirements statements and the method naming by the developers, a mapping of the two software artefacts would help developers find out the impacted software features when assessing the overall effect of the committed changes.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Fredrick Mugambi Muthengi

Department of Computing and Information Technology, Faculty of Applied Sciences, University of Embu
Embu, Kenya

Email: fremugambi@gmail.com

1. INTRODUCTION

Software maintenance involves making changes to correct errors, to modify existing features, to add new features, and to adapt to new execution platforms among others [1]. The change tasks major on updating the source code [2]. Majorly source code analysis approaches have been extensively carried out in the past years [3]–[6]. Studies have also been conducted on tracing software requirements to source code [7]–[9]. However, from the literature searches, no approach has been proposed for linking software behaviour changes to the functional requirements likely impacted by the changes made in a software system.

Functional software requirements communicate the users' needs of the software system. These are the requirements used by software engineers in the design and implementation phases. Also, software testers use them to generate test cases and test data. This demonstrates the critical role played by software requirements in the software development industry. Software behaviour is wired in the operations defined in the program. Therefore, method level source code analysis gives developers a chance to check the behavioural changes on the software away from the complexity of detailed static code analysers [10].

In many cases, system owners suggest changes in the system that are then implemented by the developers in the source code. Prior to change implementation, a change impact analysis is conducted in

which requirements play a key role. However, the one software asset that is not well updated and tracked is the software requirements artefact. This leads to code changes that may adversely deviate from the user specified expectations. Program comprehension, a precursor to making correct and effective changes in the software suffers the most in light of incomplete requirements to code traceability during software maintenance tasks.

The aim of this paper is to design and implement an approach for mapping changed methods to their associated functional requirements [11] capable of guiding developers in the assessment of the impact of changes made to the behaviour of the software system. This would help understand why the behaviour of a software system changes in particular ways after changes have been implemented and integrated in the code. Therefore, a link between the changed methods and the impacted software requirements would offer developers an opportunity to interrogate the behaviour modifications of the system without having to deal with complex analysis of the whole source code.

2. LITERATURE REVIEW

2.1. Functional requirements

Several approaches have been fronted for requirements to source code traceability [7], [8], [12]–[15]. Requirement traceability is recognised as an important aspect in the software development. Checking software completeness with regard to the software specifications requires ways to ascertain if all the requirements have been implemented. Existing approaches such as information retrieval, execution trace analysis, and static analysis have been used to create traceability links between software requirements and source code. Some approaches just create links between work items in the software project and the code during development [7].

During software maintenance, the maintenance team in the absence of traceability links between changed methods and the requirements, faces challenges in assessing the impact of applied changes in light of the affected software requirements. Inadequate, incomplete, and outdated software documentations hampers program comprehension. Therefore, a well-documented software offers a good base for understanding the system under evaluation for future upgrades [16]. In the regression testing of the applied changes, affected requirements play a definitive role in the selection and fine tuning of the tests to be run to assess the effectiveness of the changes made. A change in a software system is geared towards behaviour change or upgrade. The change in any part of the system whether a variable renaming or addition, method removal or addition, or statements changes (removal or addition) or even refactoring will eventually reflect in one or more operations defined in the system.

In test driven development, a software system functionality is derived from the user stories. These user stories that represent expected functional behaviour of the system making a requirement to be associated to one or more method definitions in the source code. This means that every defined software functionality is associated with at least one software requirement [17].

2.2. Operational changes in source code

Software changes are implemented in the source code to realise the intended behaviour. Research by Almhana *et al.* [18] an automated approach for finding and ranking the potential methods in order to localize the source of a bug based on a bug report is proposed. However, they fall short of locating the changed method that would provide developers an advantage in analysing changes for bugs reported during testing.

Developers write code in a method or create methods based on the rationale of the software requirements. The naming, the logic flow and the expected results all are guided by the user expectation as understood by the developers. Every method created and each piece of code written in a method, implements the intent of the software requirements [13].

Refactoring, a source code improvement task [19]–[25] has a key code change referred to as extract method refactoring [19]. Extract method presents a case where a method is changed by creating another method or several methods to perform some functionality previously performed by some code in the original method. The motivation being to shorten the methods and reuse some chunk of code in several methods thereby improving code maintainability. Another case of method extraction may involve code from two or more methods being taken to form one extract method. The newly extracted method will presumably be called in the methods where the codes were extracted.

2.3. Methods and requirements software artefacts

A software system consists of several artefacts: the requirements specifications, design specifications, source code, test cases and test data, among many others. A Java software system can be defined as follows:

- Software (S) consists of a set of classes (C): $S=\{C\}$
- Each class has a set of methods (M): $C=\{M\}$
- Class methods (CM): $CM=\{M1, M2, M3, M4, \dots, Mx\}$

Software S, then defined as: $S=\{C1, C2, C3, \dots, Cn\}$ where C1, C2, ...and Cn are the classes. Software consisting of a set of operations (F) implemented in the classes can be defined as: $S=\{F\}$ where $F=\{F1, F2, F3, \dots, Fi\}$ and i is the total number of operations implemented in the classes making up the software.

$$S = \sum_{i=1}^n Fi \text{ where } Fi \in F \text{ \& } i \leq n \quad (1)$$

Another definition of software as consisting of a set of software requirements denoted as R where n refers to the total number of requirements.

$$S=\{R1, R2, R3, \dots, Rn\}$$

$$S=\{r \mid r \in R\}$$

A requirement is implemented by one or more functionalities in the software. Also, a functionality could be implementing different requirements at the same time. For example:

$$R1=\{f1, f2, f3, \dots, fn\}$$

$$R2=\{f3, f4, f5, \dots, fn\}$$

$$Rk=\{fn \mid fn \in F\} \text{ where } n \text{ varies depending on number of functionalities implementing a requirement.}$$

3. METHOD

This study sampled three java projects and a software requirements dataset extracted from www.kaggle.com. The requirements dataset was edited by adding a few other requirements of concern for the sampled Java projects. The sampled projects were recently committed in git repositories and represents a realistic reflection of the actual operations in the software industry. The changed methods list was extracted from the list of methods defined in the sampled software projects. In this study, a list of changed methods was generated by random sampling from the methods defined in the projects.

3.1. Information retrieval using vector space model

Vector space model (VSM) is a widely used technique in computing textual similarity [26]. The requirements dataset consists of a collection of requirements that are treated as vectors in the common vector space. Each requirement statement is then treated as a document (vector) in the VSM. A changed method name is decomposed to its constituent words that become the terms to check in the corpus. Given the corpus dataset P containing a document (requirement) p and a term (a method name) t, the term frequency $TF(t, p)$, represents the frequency of occurrence of t in p. $DF(t, P)$ is a document frequency representing the number of documents (requirements) that contain the term t in the corpus P. A weight w of a term t in a document p from the corpus P is computed as (2) and (3):

$$IDF(t, P) = \frac{1}{DF(t, P)} \quad (2)$$

$$Wt, p, P = TF(t, p) \times IDF(t, P) \quad (3)$$

The value of the weight w signifies the number of requirements implemented by the method. The model takes into account the number of words forming the method name. Figure 1 represents the retrieval of requirements associated with a given changed method.

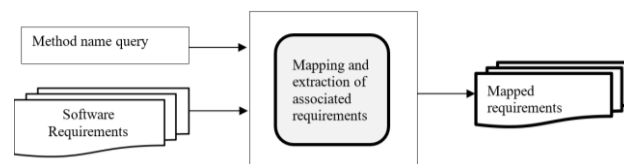


Figure 1. Retrieval of associated requirements as per the method name query

3.2. Mapping changed method (s) to requirements

This research adopted the idea that software artefacts like method names can be derived from the user requirements [27]. For example, the stories given in agile extreme programming describe software artefacts including requirements and the methods has deciphered by the developers [28]. The requirements

description languages encapsulate expected software functionalities thereby making developers derive method names from the requirements statements [29]. The method names used communicate the intent of the expressed requirement statements [13]. Hence, we extract the requirements impacted by the changes by mapping them to the method names. The investigation used the shortest name to be one word and longest method name to have four words since on average a method name consists of four words [30] for clarity in functionality representation. The study assumed best and standard practice in the software development industry in method naming conventions [31] and requirements expressions. The Algorithm 1 was used in the mapping of changed methods to the requirements affected by the method changes.

Algorithm 1

Input: (i) changed methods list
(ii) Software requirements corpus dataset
Output: Requirements associated with the changed methods
Step 1: for each method **m** in methods list **M**
Step 2: for each requirement **REQ** in requirements
Step 3: if match **m** \rightarrow **REQ**
Step 4: **mREQlist** \leftarrow Add **REQ**
Step 5: **end for**
Step 6: **end for**
Step 7: return **mREQlist**

3.3. Equivalent mapping approximations

A method name can consist of a single verb or several words describing the actual intent of the code [31]. For a one-word and two words method name a 100% match was needed to map a method name to the requirements it implements. In cases where the name contained three to six words, a 100% match was the most appropriate. The purpose of using many words in naming methods is to succinctly communicate the intention of the block of code inside it [13], [30]. Occasionally, programmers may connect several nouns in a method name. For example, the method name *searchFileOnDisk* has the word “on” which is joining the nouns file and disk to communicate the exact operation to be carried out. The actual operation is defined by the words *search* and *File*. Also, developers label identifiers with terms familiar in the application domain [32] that are used in documenting software requirements. For a one-word method name, the match would be either 100% or 0%. For a two-word method name, the match would be either 100%, 50%, or 0%. For a three-word function name, the match would be either 100%, 66.7%, 33.3%, or 0%, for a four-word method name, the match would be either 100%, 75%, 50%, 25%, or 0%. For a five-word method name, the match would be either 100%, 80%, 60%, 40%, 20%, or 0%. For a six-word method name, the match would be either 100%, 83.3%, 66.7%, 50%, 33.3%, 16.7%, or 0%. A match of $\geq 51.00\%$ guarantees a majority matching of the words forming a method name. However, as the words increase, the results reliability may dwindle hence the tuning to higher percentages like $\geq 75\%$ and 100% match. 100% match represents the best case scenario for a higher precision in the results generated. In some instances, fuzzy string matching was utilized like matching the word *add* and *additions*.

4. RESULTS AND DISCUSSION

This study sampled three Java projects. In each project, an arbitrary sample of methods to represent changed methods was used. From the sampled methods, each method was linked to the requirements it implements. The study finds that some methods may be implementing the same requirement or several requirements. The cases where a method's change impact cuts across many requirements, presents a case of a common functionality that when changed, there is a larger impact in the entire system as compared to method changes whose impact is felt in only one requirement. Tables 1 and 2 shows experiment results of the sampled changed methods associated to requirements impacted by the changes.

For a 1-word and 2-word method names, the mapping results for $\geq 51\%$, $\geq 75\%$, and 100% match are the same. This represents the best case scenario also given that by convention this is the preferred method naming. The results indicate a high relevance value. When the method name consists of three or four words, the results vary for $\geq 51\%$, $\geq 75\%$, and the 100% matchings. The 100% match produces the best relevant results with the four-word method naming reaching optimal relevance. Upon observation, methods mapping to several requirements some of which are not relevant mappings is contributed by: the requirements dataset contains requirements for several systems with differing application domains. For example, *addUser* method maps to seven requirements whilst the relevant true maps are three.

4.1. Using method names with shortened formats

Method names with initial substring like *addProd* and *addProduct*, yields the same results.

- PE: the product shall provide dynamic change support and transparent resource addition. The product shall support transparent resource addition and dynamic change support to provide scalability and avoid service interruptions of more than 1 day.
- INVENTORY: the managerial users shall add products for sales in the system.
- MN: the product shall continue to operate during upgrade change or new resource addition. The product shall be able to continue to operate with no interruption in service due to new resource additions.
- PE: izogn administrator must be able to add new products to the website within 2 minutes.
- INVENTORY: the manager shall View all existing products and add a new product detail.

Like the case shown in the methods selectTransaction and requestTicket, some requirements may cut across several systems implemented methods. This only serves to mean when a change is made say for requestTicket method, the ripple effects of the change will be spread to many requirements. This means that this change has a significant impact on the overall larger system and would therefore necessitate much more attention than other changes. Also, since the first method selectTransaction is impacting the same requirement with requestTicket method change, change impact assessment for the two methods would somewhat be limited in the same scope thereby saving on time and resources that would be expended on check each method change in isolation. We also observe that the requestTicket method could be called in many other methods implementing sections of the requirements extracted. This would mean that the change in requestTicket method has a broader effect on the entire system functionality and could be a key functionality in the system. Therefore, developers working on changing this system would devote keen attention to the operation requestTicket.

Based on the results shown in Tables 1 and 2, it is possible to map a method to the requirements it implements using the method name. One-word and two-word method names gives most relevant results at all considered percentages ($\geq 51\%$, $\geq 75\%$, and 100% matching) with a majority giving a 100% results relevance. The results show that a matching of 100% yields the best relevant results for the various method names considered. The more the words used in naming a method, the higher the precision in establishing the associated requirement. This is probably due to the fact that developers generate the many words method naming from the user functional requirements statements in an effort to document the expected operation implementing the requirement.

Table 1. Sample extract of methods mapped to requirements

Method name	Requirements associated with the method
addBook	OBS: the system admin shall be able to Add and manage books.
selectTransaction	F: the disputes system must allow the users to select disputable transactions (based on the age of the transaction) from a user interface and initiate a dispute (ticket retrieval request or chargeback notification) on the selected transaction.
requestTicket	<ul style="list-style-type: none"> – F: the disputes system will provide the user the ability to create or initiate a ticket retrieval request. As part of ticket retrieval creation process the system must prompt the user to enter all the required information to create the ticket retrieval request. The ticket retrieval request is a document that is sent to merchant inquiring the validity of a transaction. – F: the disputes system must provide a confirmation to the user upon the creation of ticket retrieval request that contains the following information; the dispute case number the type of retrieval requested (copy original or portfolio) and the date that the merchant response is due. – F: the disputes system must allow the user to create three unique types of ticket retrieval requests. The three types of ticket retrieval requests are; i) request for original receipt, ii) request for a copy of the receipt or, iii) request for a portfolio. A portfolio consists of documentation that would provide proof of a purchase such as the documentation that is received from a car rental agency that is more than a sales receipt. – F: the disputes system must allow the users to select disputable transactions (based on the age of the transaction) from a user interface and initiate a dispute (ticket retrieval request or chargeback notification) on the selected transaction. – F: the disputes system must provide search functionality. The search method must include the ability to search by; i) the dispute case number, ii) the merchant account number, iii) the card member account number, and) the issuer number. In addition to the above criteria the search functionality must further allow the user to limit the results of the search by a date range the type of dispute (ticket retrieval request or chargeback notification) the case status (open closed or all) and the dispute reason code.
Login	<ul style="list-style-type: none"> – OBS: the user shall login before performing any transaction. – BANK: the customer shall be able to sign in with login and password. – INVENTORY: it allows admin to manage two types of users, hold their details, authenticate these users at the time of login and accordingly provide different options. – INVENTORY: the system allows the godown manager to login into the system and enter their inwards entries related to their godown. – SE: the WCS system shall not allow automatic logins by any user. Cookies containing WCS login information about a user will not be stored on a user's computer. – INVENTORY: the system management users shall edit a purchase. Before a purchase edit, the user shall login to authenticate the purchase edit. – OBS: before using the system, a user shall be required to login.

Table 2. Experiment results of mapping changed methods to requirements

Index	Changed method	Requirements associated with the changed method								
		$\geq 51\%$ match			$\geq 75\%$ match			$\geq 100\%$ match		
		Extract	Actual	Precision	Extract	Actual	Precision	Extract	Actual	Precision
1	Deposit	3	3	1	3	3	1	3	3	1
2	Login	10	10	1	10	10	1	10	10	1
3	Logout	5	5	1	5	5	1	5	5	1
4	Withdraw	2	2	1	2	2	1	2	2	1
5	addBook	3	3	1	3	3	1	3	3	1
6	addCustomer	6	2	0.3333	6	2	0.3333	6	2	0.3333
7	addPurchase	1	1	1	1	1	1	1	1	1
8	addSale	3	2	0.6667	3	2	0.6667	3	2	0.6667
9	addUser	14	6	0.4286	14	6	0.4286	14	6	0.4286
10	applyLoan	1	1	1	1	1	1	1	1	1
11	availableBooks	6	6	1	6	6	1	6	6	1
12	changePassword	7	7	1	7	7	1	7	7	1
13	checkBalance	1	1	1	1	1	1	1	1	1
14	clearLoan	1	1	1	1	1	1	1	1	1
15	deleteBook	1	1	1	1	1	1	1	1	1
16	deleteProduct	3	1	0.3333	3	1	0.3333	3	1	0.3333
17	deleteSale	1	1	1	1	1	1	1	1	1
18	deleteUser	1	1	1	1	1	1	1	1	1
19	editCustomer	2	1	0.5	2	1	0.5	2	1	0.5
20	editUser	5	3	0.6	5	3	0.6	5	3	0.6
21	getAuthor	1	1	1	1	1	1	1	1	1
22	getPrice	3	3	1	3	3	1	3	3	1
23	setQuantity	1	1	1	1	1	1	1	1	1
24	updateBook	1	1	1	1	1	1	1	1	1
25	searchProduct	7	7	1	7	7	1	7	7	1
26	searchItem	1	1	1	1	1	1	1	1	1
				0.8793					0.8793	0.8793
27	searchProductName	9	1	0.1111	1	1	1	1	1	1
28	editPurchaseStock	4	3	0.75	0	0	0	0	0	0
29	getCustomerName	4	3	0.75	1	1	1	1	1	1
30	getProductInformation	16	1	0.0625	1	1	1	1	1	1
31	getProductName	12	2	0.1667	0	0	0	0	0	0
32	getSupplierInformation	3	0	0	0	0	0	0	0	0
				0.3067					0.5	0.5
33	addBookToCart	3	1	0.3333	3	1	0.3333	1	1	1
34	addItemToSale	6	1	0.1667	6	1	0.1667	1	1	1
35	removeBookFromCart	2	1	0.5	2	1	0.5	1	1	1
36	removeItemFromSale	1	1	1	1	1	1	1	1	1
				0.5					0.5	1
Overall averages				0.5620					0.6264	0.7931

In cases where the matching is capped at $\geq 50\%$, the results indicate that further processing would be needed to filter the actual relevant mappings for method names with three and above words. The main advantage of this, however, is that it ensures a majority of the matching for any number of words for a method name. Considering that the $\geq 51\%$ and $\geq 75\%$ gives results that are indicative of the method name to requirements relationship, developers can choose to configure the mapping to either $\geq 51\%$, $\geq 75\%$, or 100% depending on the level of results relevance the system provides. Tuning to 100% match would guarantee a near perfect scenario. But in cases where developers deviate a little on naming methods with regard to intent or where software requirements deviate from communicating actual intent, leaving developers to struggle in getting user's intent, the tuning of $\geq 51\%$ would be appropriate.

Also, the fact that a majority of the mapped requirements relate to the specific software methods in consideration implies that developers use domain specific terms as spelt out in the requirements. This increases the likelihood of useful mappings. In situations where developers use short names like *addProd* instead of *addProduct* the results indicate the intention of the developer and that of the user map to the same result. The results of this study agrees with the idea that methods implement the intent and expectation of the software users as detailed in the requirements specifications [13], [33].

5. CONCLUSION

Our study's objective was to establish a link between the changed methods in a particular software to the requirements whose implementations are affected by the changes made in the code. The study finds that the association between changed methods can appropriately be established with the affected functional

requirements. Since software requirements communicate the intent of the software and the developers write methods to bring out the intent, the mapping established provides a clear glimpse of the impacted functional requirements. The results are indicative of the possible impact of the method changes in the software under test. Some method changes cut across several requirements translating to several features of the system being affected making the change impact widespread. The findings also reveal that the higher the number of requirements a changed method is associated with, the higher the number of software features impacted by the change in the system. This is also a case of a common and critical operation in the system.




REFERENCES

- [1] L. Ghadhab, I. Jenhani, M. W. Mkaouer, and M. B. Messaoud, "Augmenting commit classification by using fine-grained source code changes and a pre-trained deep neural language model," *Information and Software Technology*, vol. 135, 2021, doi: 10.1016/j.infsof.2021.106566.
- [2] G. Canfora, A. D. Sorbo, S. Forootani, M. Martinez, and C. A. Visaggio, "Patchworking: exploring the code changes induced by vulnerability fixing activities," *Information and Software Technology*, vol. 142, p. 106745, 2022, doi: 10.1016/j.infsof.2021.106745.
- [3] R. Paramitha and Y. D. W. Asnar, "Static code analysis tool for laravel framework based web application," in *Proceedings of 2021 International Conference on Data and Software Engineering: Data and Software Engineering for Supporting Sustainable Development Goals, ICoDSE 2021*, pp. 1-6, 2021, doi: 10.1109/ICoDSE53690.2021.9648519.
- [4] A. Grosu, "Software tools for source code analysis," *Journal of Mobile, Embedded and Distributed Systems*, vol. 7, no. 2, pp. 47–53, 2015.
- [5] S. Wang, J. Nam, and L. Tan, "QTEP: quality-aware test case prioritization," in *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 523-534, 2017, doi: 10.1145/3106237.3106258.
- [6] R. Mudduluru and M. K. Ramanathan, "Efficient flow profiling for detecting performance bugs," in *ISSTA 2016 - Proceedings of the 25th International Symposium on Software Testing and Analysis*, pp. 413-424, 2016, doi: 10.1145/2931037.2931066.
- [7] A. Delater and B. Paech, "Tracing requirements and source code during software development: an empirical study," in *International Symposium on Empirical Software Engineering and Measurement*, 2013, doi: 10.1109/ESEM.2013.16.
- [8] P. Dai, L. Yang, Y. Wang, D. Jin, and Y. Gong, "Constructing traceability links between software requirements and source code based on neural networks," *Mathematics*, vol. 11, no. 2, pp. 315-339, 2023, doi: 10.3390/math11020315.
- [9] J. Singh, S. R. Chowdhuri, G. Bethany, and M. Gupta, "Detecting design patterns: a hybrid approach based on graph matching and static analysis," *Information Technology and Management*, vol. 23, no. 3, pp. 139–150, Sep. 2022, doi: 10.1007/s10799-021-00339-3.
- [10] R. Tufano, L. Pascarella, M. Tufano, D. Poshyanyk, and G. Bavota, "Towards automating code review activities," in *Proceedings - International Conference on Software Engineering*, no. ii, pp. 163–174, 2021, doi: 10.1109/ICSE43902.2021.00027.
- [11] K. Feichtinger, D. Hinterreiter, L. Linsbauer, H. Prähofer, and P. Grünbacher, "Guiding feature model evolution by lifting code-level dependencies," *Journal of Computer Languages*, vol. 63, no. 101034, 2021, doi: 10.1016/j.cola.2021.101034.
- [12] B. Wang, R. Peng, Y. Li, H. Lai, and Z. Wang, "Requirements traceability technologies and technology transfer decision support: A systematic review," *Journal of Systems and Software*, vol. 146, pp. 59-79, 2018, doi: 10.1016/j.jss.2018.09.001.
- [13] T. Hey, "INDIRECT: traceability-driven requirements-to-code traceability," in *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion, ICSE-Companion 2019*, pp. 190-191, 2019, doi: 10.1109/ICSE-Companion.2019.00078.
- [14] C. Strobbe, S. S. Cordero, and R. Vingerhoeds, "Open-source CubeSat MBSE approach to address traceability: power subsystem," in *SysCon 2023 - 17th Annual IEEE International Systems Conference, Proceedings*, 2023, doi: 10.1109/SysCon53073.2023.10131054.
- [15] P. Rempel and P. Mader, "Preventing defects: the impact of requirements traceability completeness on software quality," *IEEE Transactions on Software Engineering*, vol. 43, no. 8, pp. 777-797, 2017, doi: 10.1109/TSE.2016.2622264.
- [16] A. Boyarchuk, O. Pavlova, M. Bodnar, and I. Lopatto, "Approach to the analysis of software requirements specification on its structure correctness," in *CEUR Workshop Proceedings*, vol. 2623, pp. 85-95, 2020.
- [17] M. Irshad, R. Britto, and K. Petersen, "Adapting behavior driven development (BDD) for large-scale software systems," *Journal of Systems and Software*, vol. 177, no. 110944, 2021, doi: 10.1016/j.jss.2021.110944.
- [18] R. Almhana, M. Kessentini, and W. Mkaouer, "Method-level bug localization using hybrid multi-objective search," *Information and Software Technology*, vol. 131, no. 106474, 2021, doi: 10.1016/j.infsof.2020.106474.
- [19] A. Hora and R. Robbes, "Characteristics of method extractions in Java: a large scale empirical study," *Empirical Software Engineering*, vol. 25, pp. 1798-1833, 2020, doi: 10.1007/s10664-020-09809-8.
- [20] C. Ragkhitwetsagul, J. Krinke, and D. Clark, "A comparison of code similarity analysers," *Empirical Software Engineering*, no. 23, pp. 2464-2519, 2018, doi: 10.1007/s10664-017-9564-7.
- [21] J. Pantiuchina, B. Lin, F. Zampetti, M. D. Penta, M. Lanza, and G. Bavota, "Why do developers reject refactorings in open-source projects?," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 2, pp. 1-3, 2022, doi: 10.1145/3487062.
- [22] Y. Zhang, C. Li, and Y. Bai, "Consistency validation method for Java fine-grained lock refactoring," *IEEE Access*, vol. 9, pp. 149287-149301, 2021, doi: 10.1109/ACCESS.2021.3120414.
- [23] A. Brito, A. Hora, and M. T. Valente, "Towards a catalog of composite refactorings," *Journal of Software: Evolution and Process*, vol. 36, no. 4, Jan. 2024, doi: 10.1002/smr.2530.
- [24] I. H. Moghadam, M. Ó. Cinnéide, A. Sardarian, and F. Zarepour, "Model-based source code refactoring with interaction and visual cues," *Journal of Software: Evolution and Process*, no. e2596, 2023, doi: 10.1002/smr.2596.
- [25] V. Alizadeh, M. Kessentini, M. W. Mkaouer, M. Ocinneide, A. Ouni, and Y. Cai, "An interactive and dynamic search-based approach to software refactoring recommendations," *IEEE Transactions on Software Engineering*, vol. 46, no. 9, pp. 932-961, 2020, doi: 10.1109/TSE.2018.2872711.
- [26] M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild: automatically augmenting incomplete trace links," in *Proceedings - International Conference on Software Engineering*, 2018, pp. 834–845. doi: 10.1145/3180155.3180207.
- [27] F. Dalpiaz, P. Gieske, and A. Sturm, "On deriving conceptual models from user requirements: an empirical study," *Information*




- and *Software Technology*, vol. 131, no. 106484, 2021, doi: 10.1016/j.infsof.2020.106484.
- [28] J. Medeiros, A. Vasconcelos, C. Silva, and M. Goulão, "Requirements specification for developers in agile projects: evaluation by two industrial case studies," *Information and Software Technology*, vol. 117, pp. 1-21, 2020, doi: 10.1016/j.infsof.2019.106194.
- [29] A. Ohnishi, "Software requirements specification database based on requirements frame model," in *Proceedings of the IEEE International Conference on Requirements Engineering*, pp. 221-228, 1996, doi: 10.1109/ICRE.1996.491450.
- [30] R. Alsuhbani, C. Newman, M. Decker, M. Collard, and J. Maletic, "On the naming of methods: a survey of professional developers," *Proceedings-International Conference on Software Engineering*, 2021, pp. 587-599, doi: 10.1109/ICSE43902.2021.00061.
- [31] "Naming Conventions," Oracle, 1999, [Online]. Available: <https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html>. (Accessed: Feb. 10, 2024).
- [32] A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Labeling source code with information retrieval methods: An empirical study," *Empirical Software Engineering*, vol. 19, no. 5, pp. 1383-1420, 2014, doi: 10.1007/s10664-013-9285-5.
- [33] H. Cibulski and A. Yehudai, "Regression test selection techniques for test-driven development," in *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011*, 2011, pp. 115-124, doi: 10.1109/ICSTW.2011.28.

BIOGRAPHIES OF AUTHORS






Fredrick Mugambi Muthengi    holds a master of science degree in computer Science from the University of Hull, UK and a bachelor of science in computer science from Masinde Muliro University of Science and Technology, Kenya. He is Ph.D. student in computer science programme at the University of Embu. His research interests are software regression testing, maintaining large software systems, and artificial intelligence in education. He can be contacted at email: fremugambi@gmail.com.






Dr. David Muchangi Mugo    is a Ph.D. holder of information systems from Kenyatta University, Kenya. He has a master of science degree in computer science from Technical University of Hamburg, Germany and a Masters of Business Administration where he specialized in Technology Management from Northern Institute of Technology Management, Hamburg, Germany. He graduated with a first class honors degree in bachelor of science in computer science from Kenyatta University. His research interests include ICT for development, electronic health and deployment of artificial intelligence to transform agricultural and health sector. He can be contacted at email: david.mugo@embuni.ac.ke.



Dr. Stephen Makau Mutua    holds Ph.D. in systems analysis and integration, a master in information technology and bachelor of science in computer science. He is an Associate Professor in the Department of Computer Science in Meru University of Science and Technology and currently serving as the dean of the School of Computing and Informatics. He is an established scholar and academician with several publications in refereed journals and book chapters. His research interests are neural networks, computer networks, and data science. He can be contacted at email: smutua@must.ac.ke.



Dr. Faith Mueni Musyoka    is a Lecturer in the Department of Computing and Information Technology at the University of Embu, Kenya. She possesses Ph.D. in information technology from Kabarak University, Kenya, M.Sc. information technology and B.Sc. computer science both from Masinde Muliro University of Science and Technology. She is an esteemed member of ACM and OWSD, and has extensive list of publications in well-regarded journals. Her research interests encompass a wide spectrum, from software quality metrics to health informatics. She can be contacted at email: mueni.faith@embuni.ac.ke.